Software-Projekt I

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik
Fachbereich Mathematik und Informatik
Universität Bremen

Sommersemester 2013

Objektorientierte Modellierung I

Objektorientierte Modellierung

Fragen

- Was ist Modellierung und wozu brauchen wir sie?
- Wie wird objektorientiert modelliert?
- Wie kann man die Unified Modeling Language (UML) für die Modellierung verwenden?

Anmerkung: kein umfassender UML-Kurs; zur UML siehe z.B.: Störrle (2005); Rupp u. a. (2007).

Modellierung

- Was ist ein Modell?
  - Abbild eines Originals
- Wozu modellieren wir?
  - um etwas zu verstehen
  - um Vorhersagen machen zu können
  - um etwas zu dokumentieren
- Wann modellieren wir?
  - jederzeit: Projektplan, Anforderungen, Architektur, ...

Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen zwischen Objekten
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Dies ist keine streng sequenzielle Folge. Die Aktivitäten verlaufen vielmehr parallel und iterativ.

Ist → Soll

Schwierig: Dinge im Abstrakten beschreiben.

Einfacher: von konkreten Geschäftsprozessen ausgehen.

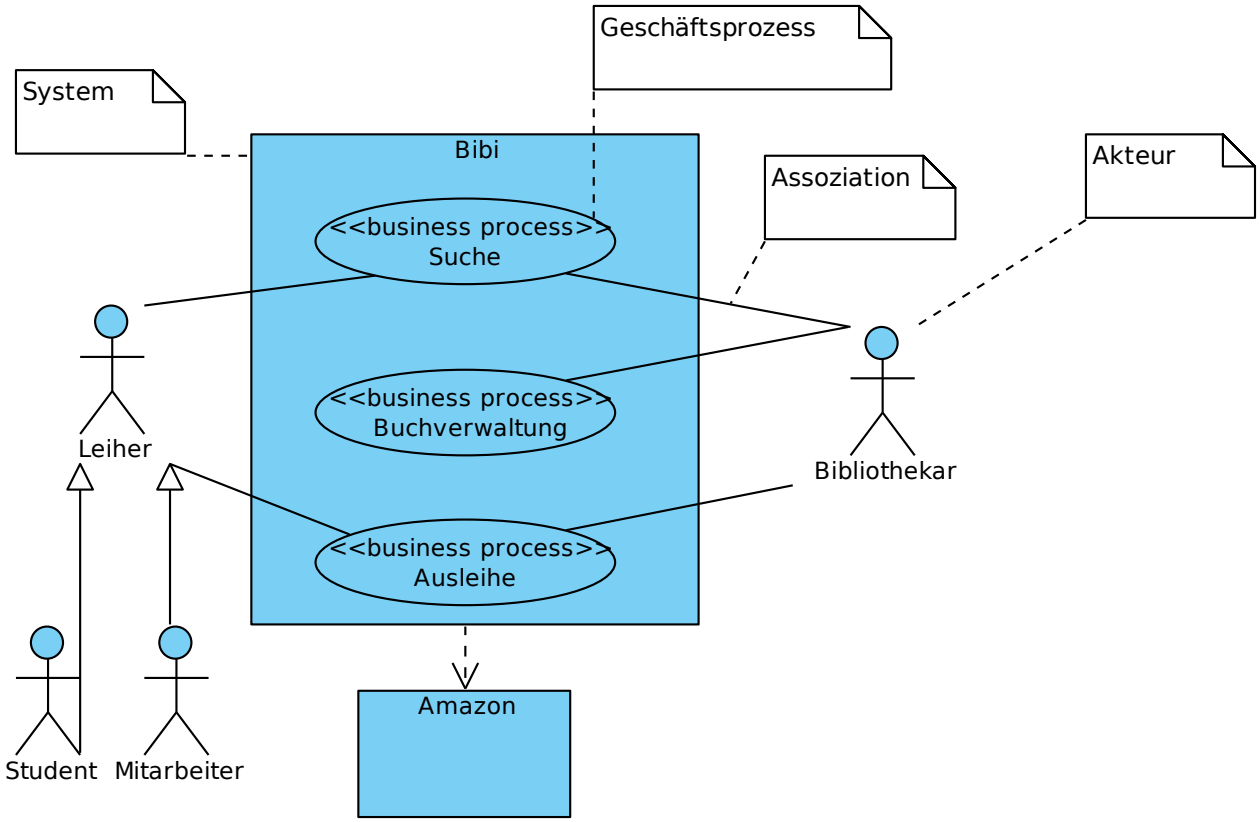
Definition

Ein **Geschäftsprozess** ist eine Folge von Schritten oder ein Rezept, um ein Geschäftsresultat zu erzielen.

Beispiele:

- Ausleihe: Vormerkung, Abholung, Mahnung, Rückgabe
- Buchverwaltung: Anschaffung neuer Bücher, Ausinventarisierung
- Buchsuche

Geschäftsprozesse in UML (OMG)



In UML gibt es keinen notationellen Unterschied zwischen Geschäftsprozessen und Anwendungsfällen (siehe später). Deshalb werden Geschäftsprozesse hier durch einen eigenen Stereotyp *business process* gekennzeichnet.

Assoziationen werden in Anwendungsfalldiagrammen nicht benannt. Assoziationen sind keine Datenflüsse; Assoziationen beschreiben Kommunikationsbeziehungen zwischen Akteuren und dem System.

Akteur

Definition
Akteur

- repräsentiert eine kohärente Menge von Rollen, die von Benutzern in der Interaktion mit dem System eingenommen werden können
- können Menschen und andere Dinge sein (z.B. andere automatisierte Systeme)

Beispiel rechnergestützter Geschäftsprozess *Ausleihe* in der Bibliothek:

- Buchsuche: Mitarbeiter sucht ein Buch
  → Akteure: Mitarbeiter, System
- Abholung: Bibliothekar händigt Buch aus und vermerkt die Ausleihe im System
  → Akteure: Bibliothekar, System

Geschäftsprozess und Anwendungsfall (Use-Case)

Merkmale von Geschäftsprozessen:

- systemübergreifend,
- unterbrechbar,
- lang laufend,
- erfordern fortlaufende Interaktion zwischen vielen Akteuren,
- bestehen aus Anwendungsfällen.

Definition
Anwendungsfall (auch: Nutzfall)

- beschreibt eine Menge von Aktionssequenzen (Varianten eingeschlossen)
- jede Sequenz repräsentiert die Interaktion zwischen externen Akteuren mit dem System
- Folge ist beobachtbares Resultat, relevant für Akteur

Textuelle Beschreibung von Anwendungsfällen I

- Name: Vormerkung eines Buches
- Akteure:
  - Leiher
- Vorbedingung:
  - Leiher möchte Buch ausleihen
  - Leiher hat ein Konto in der Bibliothek
  - Rechner von Leiher hat Verbindung zu Bibliotheks-Server
- Nachbedingung:
  - Leiher hat Buch vorgemerkt
- Ablauf:
  1. Mitarbeiter startet Client und meldet sich beim Bibliotheks-Server an
  2. Leiher gibt Suchkriterium an (z.B. Name des Autoren)
  3. Server liefert alle passenden Bücher
  4. Mitarbeiter wählt Buch aus und markiert es zur Vormerkung

Textuelle Beschreibung von Anwendungsfällen II

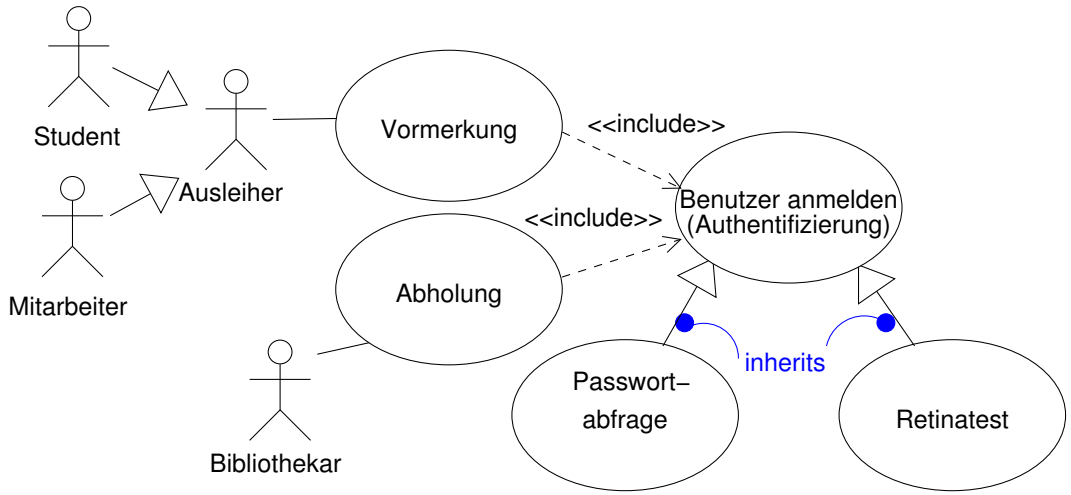
- Varianten:
  - Leiher findet kein passendes Buch
    → Leiher verändert Suchkriterium bzw. gibt Suche auf
  - Leiher hat Passwort vergessen
    → System schickt das Passwort an abgespeicherte E-Mail-Adresse
  - Buch ist bereits mehrfach vorgemerkt
    → Leiher verzichtet auf Vormerkung
  - keine Verbindung zwischen Client und Server
    → Leiher versucht es später wieder

Geschäftsprozesse, Anwendungsfälle und Akteure

1. Bestimme Geschäftsprozesse
2. Identifiziere Akteure
3. Betrachte System aus der Sicht der Akteure
4. Bestimme Anwendungsfälle für Akteure
   → liefert möglicherweise neue Akteure
5. zurück zu 2, bis keine neuen Akteure/Anwendungsfälle mehr gefunden werden können
6. identifiziere gemeinsame Anteile in Anwendungsfällen und faktorisiere entsprechend
7. fasse ähnliche Anwendungsfälle und Akteure in Vererbungshierarchien zusammen

Leider gibt es in der UML keine direkte visuelle Unterscheidung zwischen Geschäftsprozessen und Anwendungsfällen. Beide werden mit der gleichen Ellipsenform dargestellt. Aus diesem Grund sind die Geschäftsprozesse in diesem Diagramm dunkelblau gezeichnet. Alternativ kann man den Stereotyp *business process* für Geschäftsprozesse verwenden (siehe oben).

Strukturierungskonzepte für Anwendungsfälle (OMG)



Mit Hilfe von <<include>> kann man gemeinsames Verhalten aus Anwendungsfällen herausfaktorisieren und nur einmal beschreiben; hilft, Redundanz zu vermeiden.

Bedeutung B <<include>> A:

- Anwendungsfall B (Base) inkludiert das Verhalten von Anwendungsfall A (Inklusion)
- Anwendungsfall B ist ohne Inklusion A nicht notwendigerweise ein vollständiger Anwendungsfall
- Inklusion A kann ein Fragment sein

Verwendung

- Verhalten von Inklusion A wird in mehreren Anwendungsfällen benötigt

Generalisierung: Anwendungsfälle können durch die inherit-Beziehung spezialisiert werden. Entspricht der objektorientierten Vererbung.

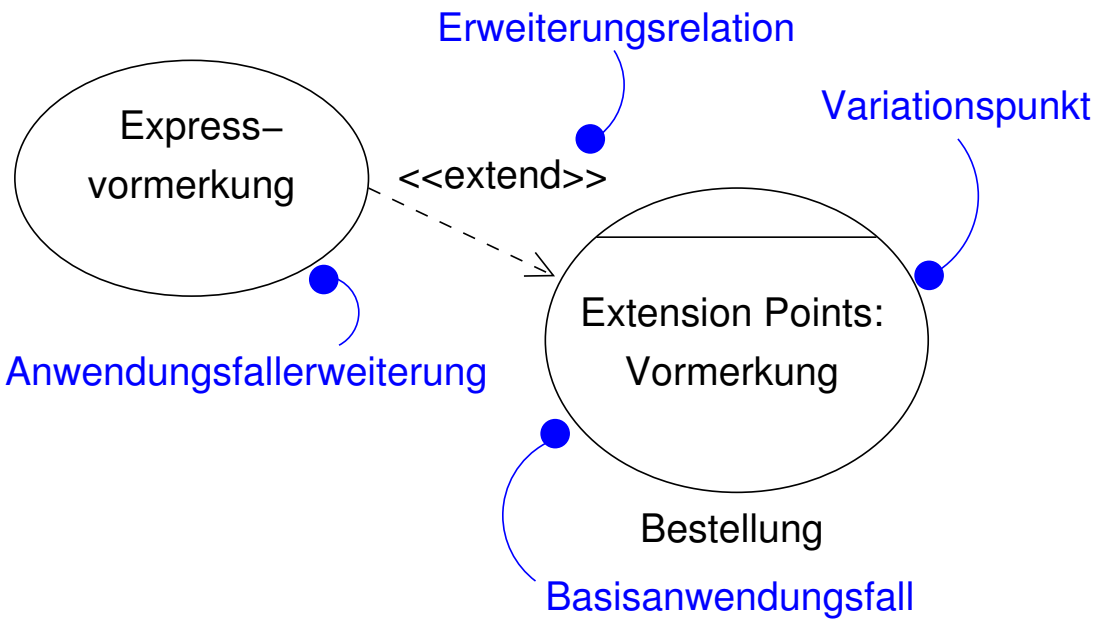
Bedeutung

- Anwendungsfall A (Child) erbt das Verhalten von Anwendungsfall B (Parent)
- Anwendungsfall B ist ein vollständiger, von A unabhängiger Anwendungsfall
- Anwendungsfall A ist ein vollständiger Anwendungsfall

Verwendung

- Verhalten eines Anwendungsfalls wird spezialisiert

Strukturierungskonzepte für Anwendungsfälle (OMG)



Die Extend-Relation erlaubt die Beschreibung optionalen Verhaltens. Erlaubt damit die Trennung des optionalen vom notwendigen Verhalten. Kann auch benutzt werden, um ein alternatives Teilverhalten zu beschreiben, das an Bedingungen geknüpft ist (z.B. Fehlerausgabe und Benachrichtigung des Systemadministrators falls Passwort falsch ist). Kann auch verwendet werden, um Teilsequenzen zu kombinieren, deren Ausführung vom Benutzer bestimmt wird (z.B. Dialog zur Auswahl von Operationen).

Bedeutung: A <<extend>> B:

- Anwendungsfall A (Extension) erweitert das Verhalten von Anwendungsfall B (Base)
- Anwendungsfall B ist auch ohne Extension A ein vollständiger Anwendungsfall
- Extension A kann ein Fragment sein

Verwendung

- Verhalten eines Anwendungsfalls wird erweitert (Spezialfall mit vorgesehener Erweiterung)

Beispiel hier: ist ein zu bestellendes Buch als Express vorgemerkt, dann wird es per Express bestellt.

Abgrenzung zur Generalisierungsbeziehung: Die extends-Relation erweitert das Verhalten an spezifischen Stellen. Der Basisanwendungsfall ist für sich ein gültiger Anwendungsfall. Durch Generalisierungsbeziehung wird das Verhalten des Basisanwendungsfalls vererbt und kann vollständig verändert werden.

Abgrenzung zur include-Relation: include schließt das Verhalten immer mit ein.

Beziehung <<extend>>

Definition

Extension Point: Ort im Verhalten eines Anwendungsfalls, an dem das Verhalten durch anderen Anwendungsfall erweitert werden kann.

Definition

A <<extend>> B: beschreibt, wann und wie das Verhalten im erweiternden Anwendungsfall A in den erweiterten Anwendungsfall B eingefügt werden kann.

Beschreibung von Anwendungsfällen

Anwendungsfalldiagramme:

- deklarieren voneinander unabhängige Anwendungsfälle
- beschreiben die Einbettung der Anwendungsfälle in den Systemkontext (externe Akteure)
- beschreiben (konkretisieren) die Anwendungsfälle jedoch nicht
  → folgt später
- sind Ausgangspunkt für die objektorientierte Modellierung
  - statische Eigenschaften (Attribute)
  - dynamische Eigenschaften (Verhalten)

Beispiel Bibliothek

Identifiziere Objekte: Suche nach Substantiven in Anwendungsfällen.

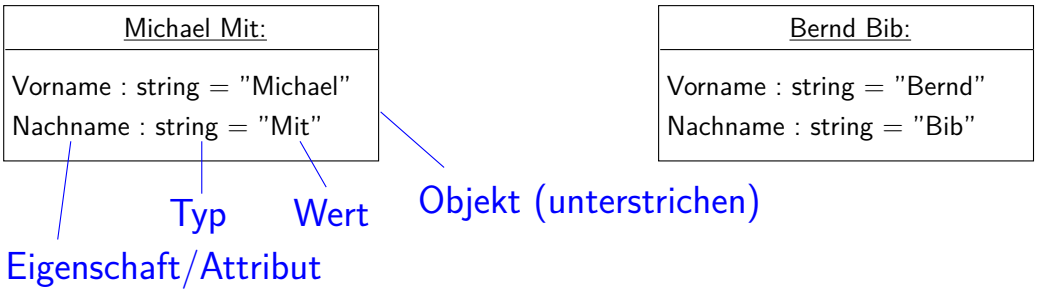
Operation: Vormerkung und Abholung

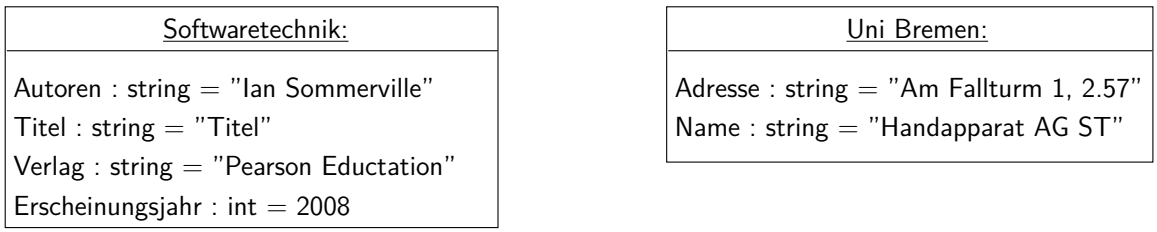
- **Mitarbeiter *Michel Mit*** möchte **Buch** ausleihen
- *Michel Mit* sucht online nach einem Buch mit **Titel** *Softwaretechnik*
- *Michel Mit* reserviert das gefundene Buch
- *Michel Mit* geht zum **Bibliothekar *Bernd Bib*** der **Universität Bremen**
- *Bernd Bib* sucht nach **Reservierung**
- *Bernd Bib* hält **Ausleihe** fest
- *Michel Mit* geht mit Buch von dannen

Reservierung und Ausleihe sind substantivierte Verben. In unserem Falle betrachten wir diese Substantive nicht wirklich als Substantive. Sie werden später als Beziehungen modelliert.

Anwendungsfall → Objekte

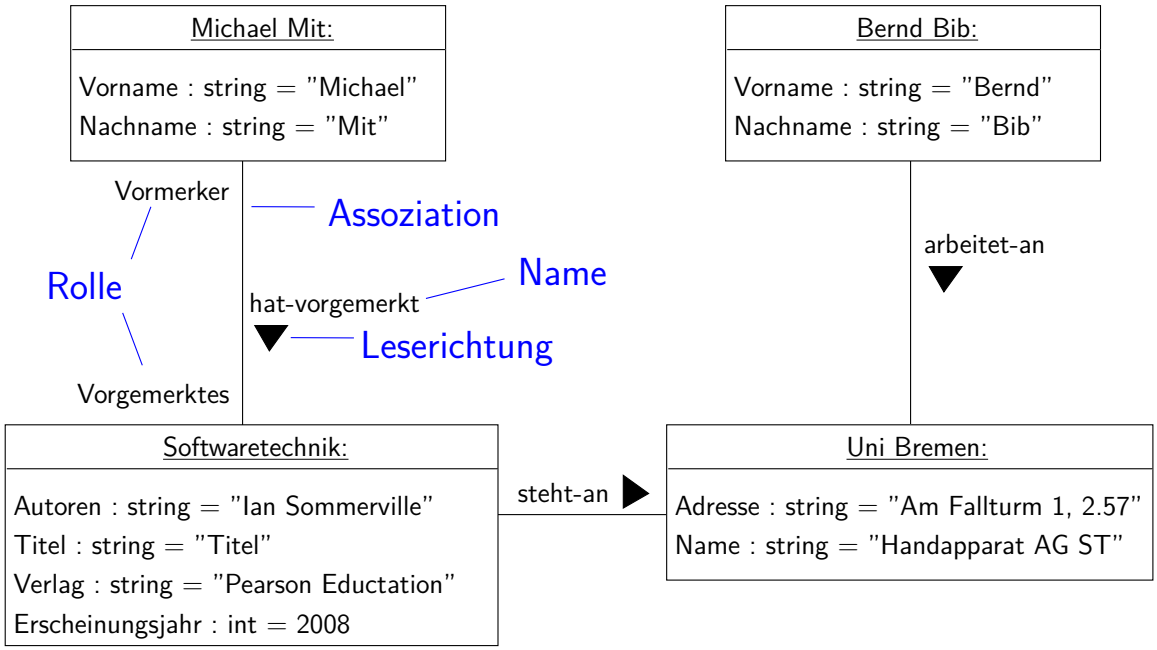
Objekt: „Ding“ (Gegenstand, Entität) mit eigener Identität





Anwendungsfall → Objekte

Assoziation: Beziehung zwischen Objekten



Objekte versus Klassen

Instanz-Ebene

- **Objektdiagramme** beschreiben statische Zusammenhänge auf der Ebene einzelner, konkreter Dinge

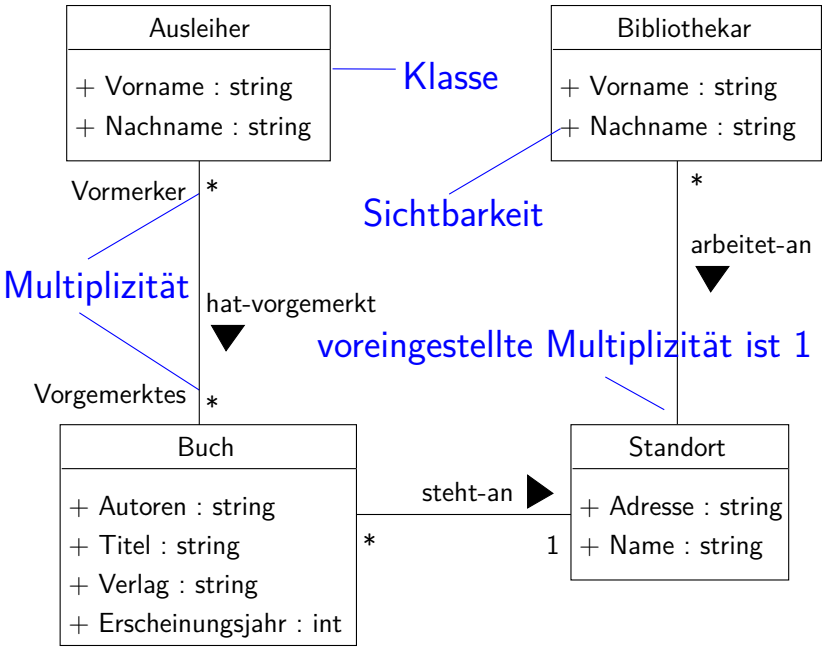
Schema-Ebene

- **Klassendiagramme** beschreiben statische Zusammenhänge unabhängig von Details konkreter Objekte, auf der Ebene mehrerer gleichartiger Dinge

Bemerkung: Objektdiagramme müssen konform zu den Klassendiagrammen sein.

Objekte → Klassen

Klasse: Menge von gleichartigen Objekten mit gemeinsamen Eigenschaften



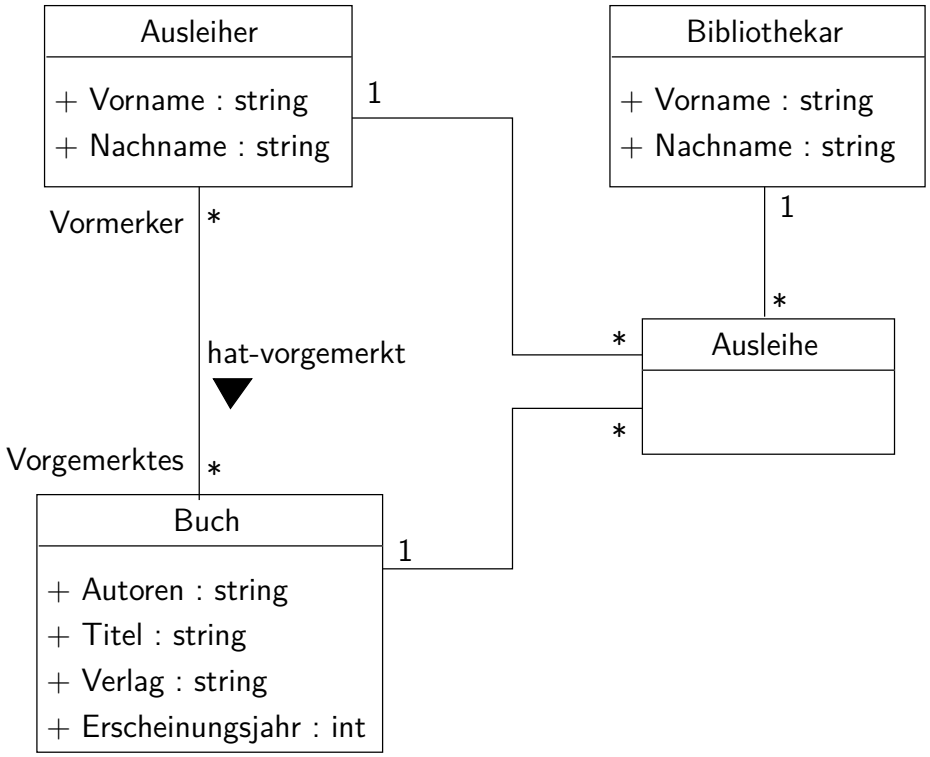
Die Sichtbarkeit von Attributen in der UML wird wie folgt repräsentiert:

- \+ öffentlich (public)
- – versteckt (private)
- # sichtbar für Unterklasen (protected)

Da wir hier Anwendungskonzepte modellieren, sollen alle Attribute öffentlich sichtbar sein. Ansonsten gäbe es keinen Grund, sie hier aufzuführen. In Klassendiagrammen, die für den Entwurf und die Implementierung benutzt werden, sollten Attribute in der Regel versteckt werden.

Objekte → Klassen: Assoziationen

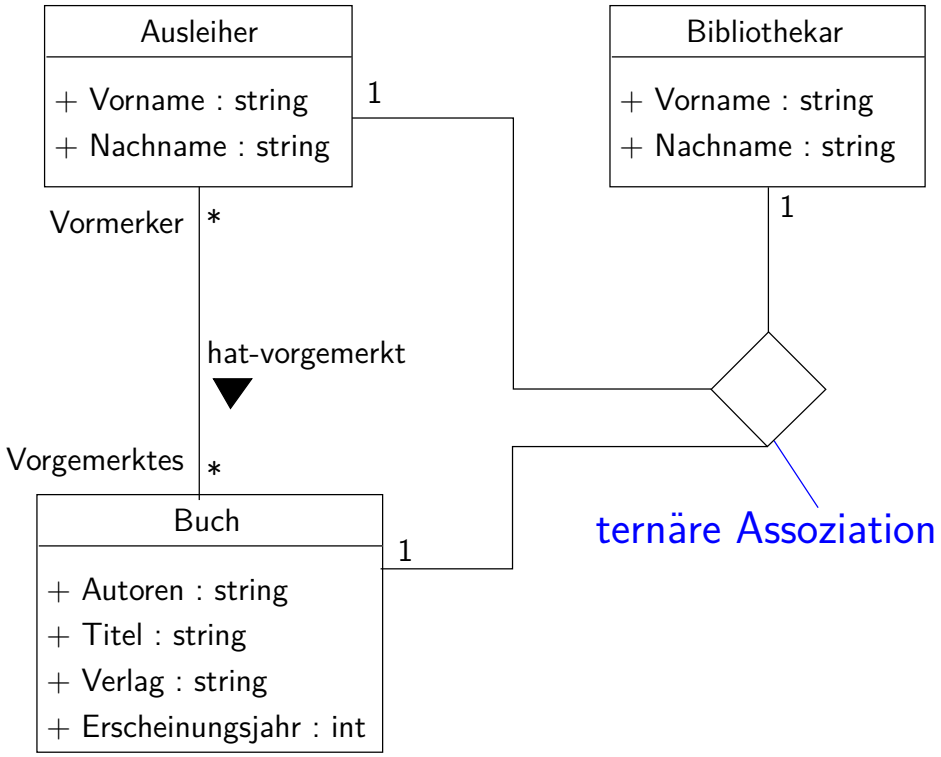
mehrstellige Assoziation als Klasse:



Die bisher vorgestellten Assoziationen können maximal zwei verschiedene Klassen verbinden. Sollten mehrstellige Assoziationen modelliert werden, könnte man dafür Klassen verbinden. Allerdings ist eine Assoziation eigentlich eine Relation. Eine Relation erkennt man daran, dass sie ohne die beteiligten Objekte (die Instanzen der Klassen, die mit der Assoziation verbunden sind) nicht existieren kann. Zu jeder Ausleihe müssen vorgemerktes Buch und vormerkender Ausleiher existieren. Verwendet man gewöhnliche Klassen zur Modellierung mehrstelliger Assoziation geht dieser semantische Unterschied verloren.

Objekte → Klassen: Assoziationen

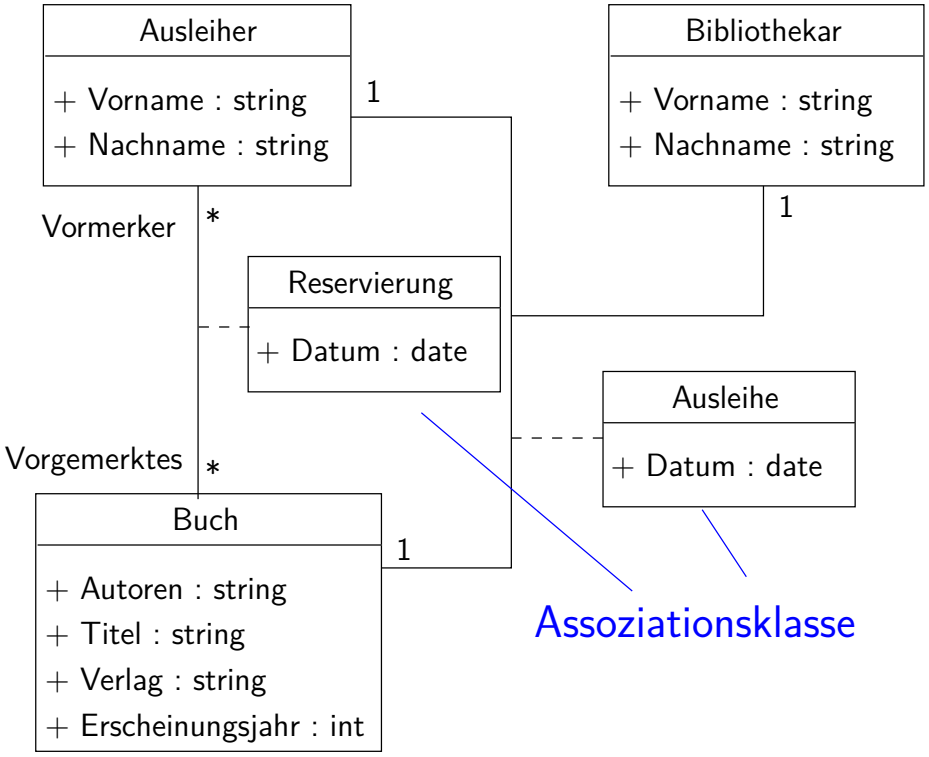
mehrstellige Assoziation:



Mehrstellige Assoziationen (d.h. Assoziationen zwischen mehr als zwei Klassen) können besser mittels mehrstelligen Assoziationen dargestellt werden. Hierfür verwendet man eine Rautendarstellung (bekannt auch aus so genannten Entity-Relationship-Diagrammen, die Vorbilder der Klassendiagrammen waren).

Objekte → Klassen: Assoziationen

(mehrstellige) Assoziation mit Attributen als Assoziationsklasse:



Mehrstellige Assoziationen können auch mittels so genannter Assoziationsklassen modelliert werden. Dies wird man insbesondere dann machen, wenn man Assoziationen mit Attributen versehen möchte. Hier hat die binäre Reservierung ein Attribut *Datum*. Die mehrstellige Assoziation *Ausleihe* hat das gleiche Attribut und verbindet gleich drei Klassen.

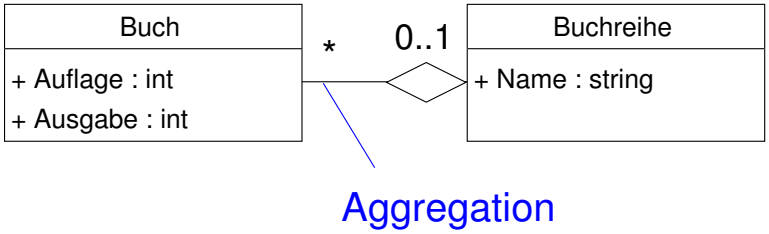
Merke: Assoziationsklassen

- beschreiben Beziehungen, die gleichzeitig Klassen sind
- werden genutzt, um Assoziationen Attribute zuzuordnen
- werden genutzt, um mehrere Klassen in Beziehung zu setzen
- können eigene Beziehungen eingehen

Definition

Aggregation

- ist spezielle Assoziation zur Verdeutlichung von „Teil-Ganzes-Beziehungen“
- beschreibt Zusammenfassung von Komponenten zu einem Aggregat

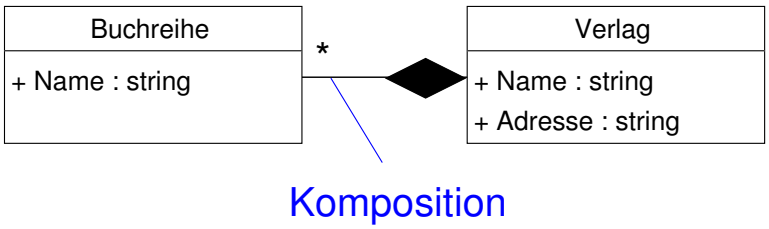


Leserichtung der Aggregation: Eine Buchreihe besteht Büchern; ein Buch kann Teil höchstens einer Buchreihe sein.

Definition

Komposition ist spezielle Aggregation:

- Existenz der Komponenten ist an die Existenz des Aggregats gekoppelt,
- jede Komponente gehört zu genau einem Aggregat (strong ownership)



Eine Buchreihe ist ohne Verlag nicht vorstellbar und eine Buchreihe gehört immer genau zu einem Verlag, darum könnte man eine Buchreihe als Komposition modellieren.

Andererseits ist eine Buchreihe höchsten logisch Teil eines Verlages, d.h. nicht im physischen Sinne. Man sollte also das Verhältnis zwischen Buchreihe und Verlag besser als gewöhnliche Assoziation modellieren.

Generalisierungen

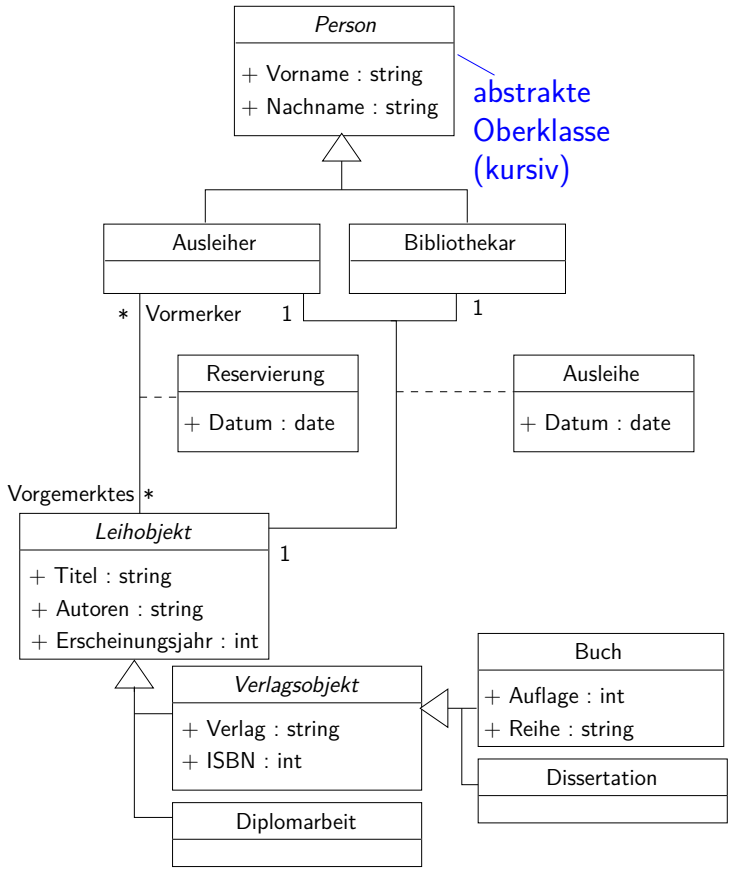
- sind spezielle Beziehung auf Schema-Ebene
- beschreiben Beziehungen zwischen einer allgemeineren Klasse (Oberklasse, Superclass) und einer spezielleren Klasse (Unterklasse, Subclass)
- die Unterklasse „erbt“ die Eigenschaften der Oberklasse:
  - Attribute
  - Methoden und deren Aufrufschnittstellen
  - Assoziationen

Unterklassen können

- neue Attribute, Methoden und Assoziationen definieren
- ererbte Attribute, Methoden und Assoziationen redefinieren (überschreiben)
- von mehreren Oberklassen erben (Mehrfachvererbung)

Bemerkung: unterschiedliche Umsetzung in verschiedenen Programmierumgebungen bzgl. Überschreiben und Mehrfachvererbung

Klassen → Klassenhierarchien



Gemeinsame Attribute werden in gemeinsamen Oberklassen zusammengefasst. *Ausleiher* und *Bibliothekar* haben jeweils Namen. Dies kann besser modelliert werden, indem *Ausleiher* und *Bibliothekar* von einer gemeinsamen abstrakten Oberklasse *Person* erben, die die gemeinsamen Attribute deklariert. Neben verlegten Lehrbüchern hat die Bibliothek auch Dissertationen und Diplomarbeiten. Alle können ausgeliehen werden und haben Attribute gemeinsam. Auch dies sollte explizit modelliert werden.

Bei der Modellierung von Anwendungskonzepten kann es gelegentlich zu Mehrfachvererbungen kommen. Dies ist nicht weiter tragisch, solange in jedem Falle das Liskovsche Substitutionsprinzip erfüllt ist (siehe unten). Bei der Abbildung des Analysemodells auf eine konkrete Implementierung mit einer Programmiersprache ohne Mehrfachvererbung wird es dann allerdings zu einem Bruch kommen. In der Analysephase konzentrieren wir uns jedoch auf eine genaue und einfache Modellierung und sollten uns deshalb von solchen Implementierungseinschränkungen noch nicht beirren lassen.

Meist sind diese neuen Oberklassen abstrakt, d.h. es gibt eigentlich kein Objekt, das genau von diesem Typ ist. So ist *Person* natürlich ein abstraktes Gebilde: Es gibt eigentlich nur Ausleiher oder Bibliothekare, aber nichts, was man nur als Person bezeichnen würde.

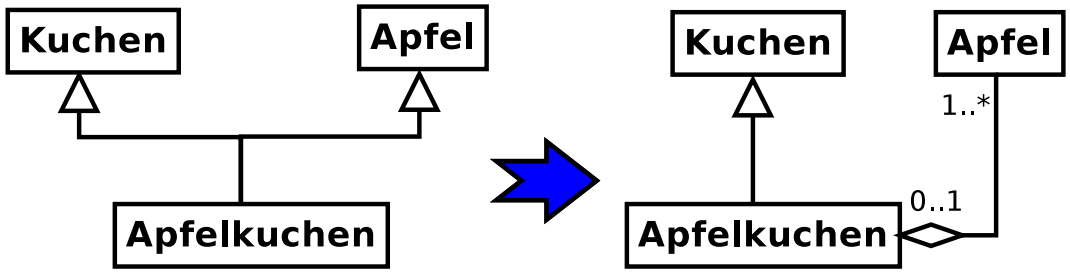
Abstrakte Klassen sind daran zu erkennen, dass ihr Name kursiv geschrieben wird. In diesem Beispiel sind *Person*, *Leihobjekt* und *Verlagsobjekt* abstrakt. Alternativ kann mit abstrakte Klassen mit dem Stereotyp *abstract* versehen.

Liskovs Substitutionsprinzip (1988; 1994)

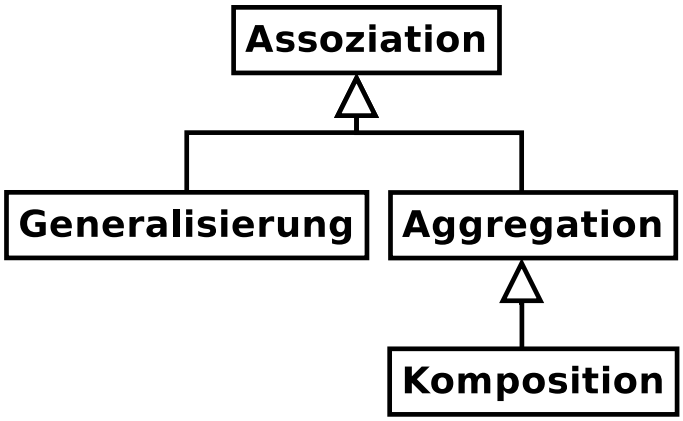
Wie kann man entscheiden, ob eine Klasse eine Spezialisierung einer anderen Klasse ist?

Definition

Liskovs Substitutionsprinzip: jede Instanz der Unterklasse kann immer und überall dort eingesetzt werden, wo Instanzen der Oberklasse auftreten.



Vergleich der Assoziationstypen



- Vererbung: ist-ein-Relation (Liskovs Substitutionsprinzip erfüllt)
- Aggregation: teil-von-Relation
- Komposition: teil-von-Relation
  - Teil gehört zu genau einem Ganzen
  - Teil existiert nur im Kontext des Ganzen
- allgemeine Assoziation: sonst

Zusammenfassung Klassendiagramme

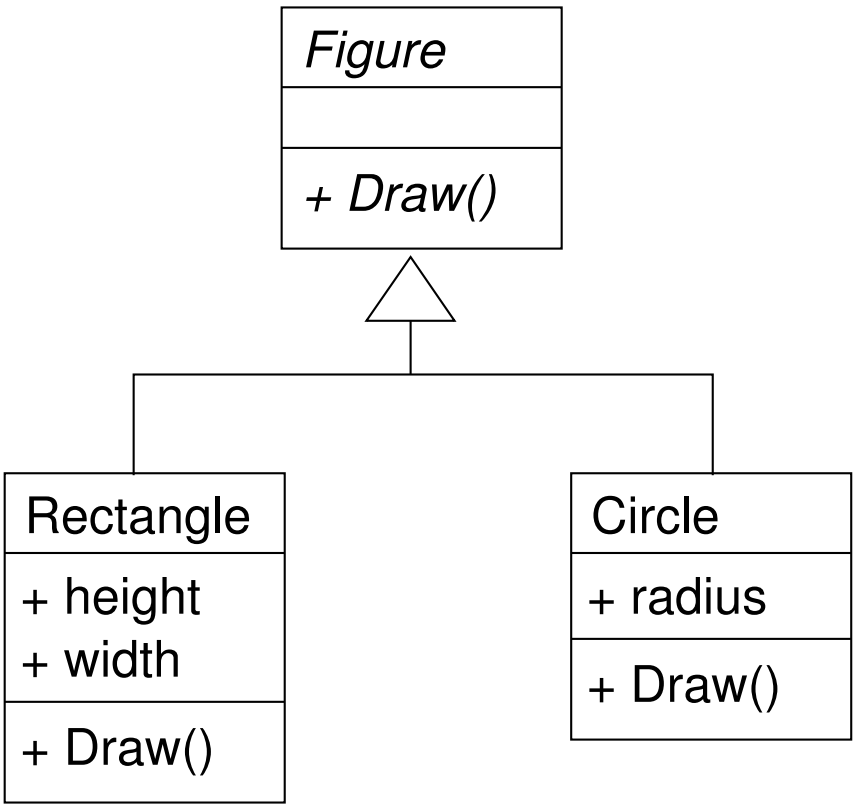
Beschreibungsinhalt

- statische Systemaspekte
- Beschreibung der wesentlichen, unterscheidbaren Dinge eines Systems und ihrer Beziehungen

zentrale Modellierungskonstrukte:

- Klassen
- Assoziationen (Beziehungsklassen)
  - spezielle Assoziationen: Generalisierung und Aggregation/Komposition

Klassendiagramme im Entwurf (statt Datenmodellierung)



Eine offene Schnittstelle

```
abstract class Figure {
    public abstract void draw ();
}

class Circle extends Figure {
    public void draw () {};
    public int radius;
}

class Rectangle extends Figure {
    public void draw () {};
    public int height;
    public int width;
}
```


Geheimnisprinzip (Information Hiding) nach Parnas (1972)

Schnittstellen sind ein Kontrakt zwischen:

- Verwender:
  - darf sich nur auf zugesicherte Annahmen verlassen
  - muss Vorbedingungen einhalten
- Anbieter:
  - muss zugesichertes Verhalten implementieren
  - darf sich nur auf zugesicherte Vorbedingungen verlassen

Der Kontrakt führt zu einer Kopplung zwischen Verwender und Anbieter.

Schnittstellen werden so entworfen, dass

- Kopplung auf das Mindestmaß beschränkt wird;
- d.h. die Details, die sich ändern können, werden hinter Schnittstelle verborgen.

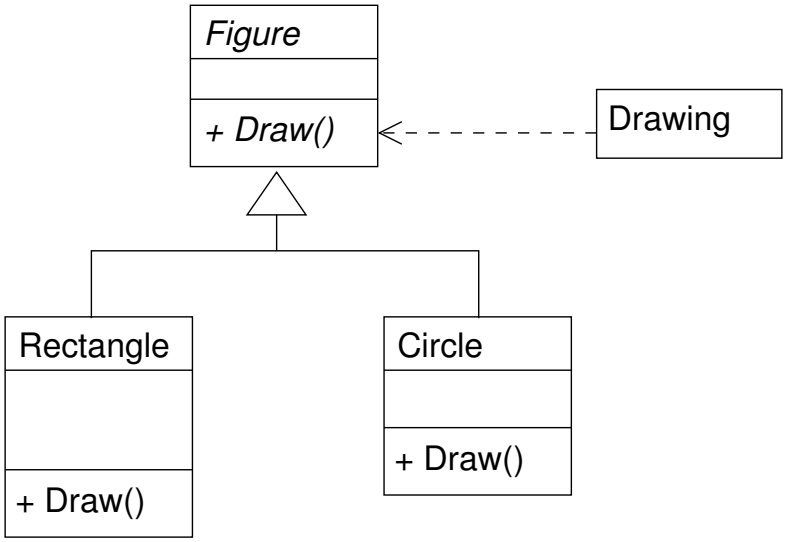
Programmiersprachenunterstützung

```
abstract class Figure {

    public abstract void draw ();
}
class Circle extends Figure {

    public void draw () {};
    private int radius;
}
class Rectangle extends Figure {

    public void draw () {};
    private int height;
    private int width;
}
```

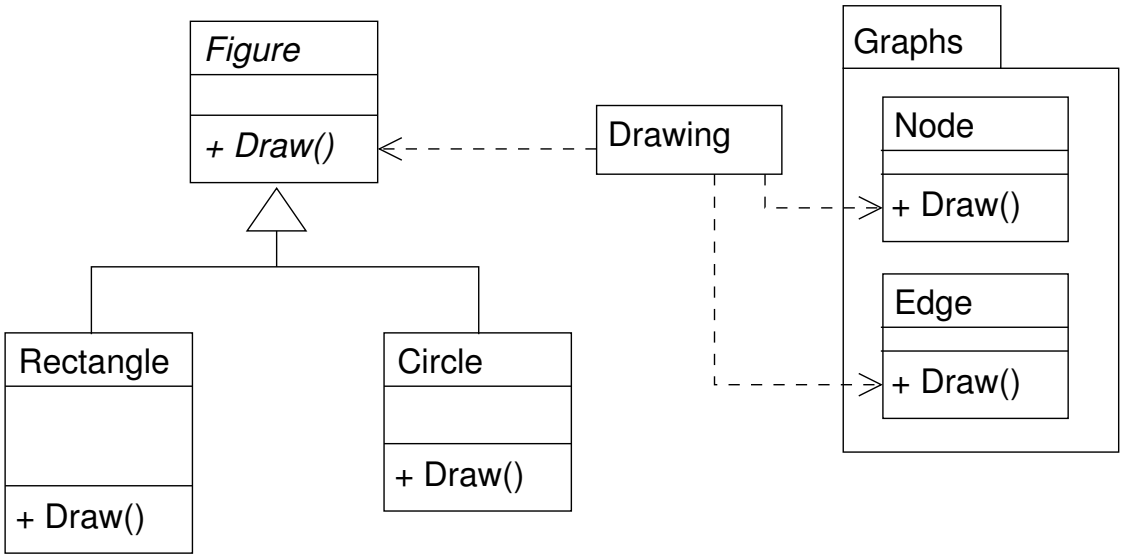

Verwender der Klasse

```
public void Drawing (Figure aFigure, int times) {
  for (int i = 0; i < times; i++) {
     aFigure.draw ();
  }
}
```

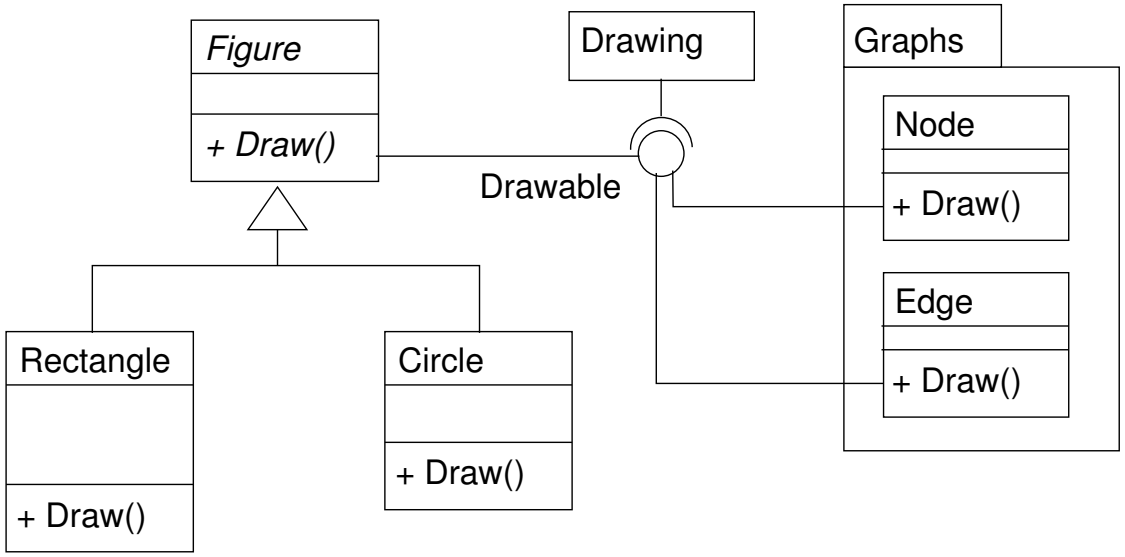
Klassendiagramm



Klassendiagramm



Klassendiagramm



Schnittstelle als eigenes Sprachkonstrukt

Schnittstelle:

```
interface Drawable {
    void draw ();
}
```

Schnittstellenverwender:

```
public void Drawing (Drawable drawableObject, int times) {
  for (int i = 0; i < times; i++) {
     drawableObject.draw ();
  }
}
```

Schnittstellenanbieter:

```
abstract class Figure implements Drawable {}
```

Zusammenfassung zu Schnittstellen

- Schnittstelle ist Kontrakt zwischen Anbieter und Verwender, der die erlaubten wechselseitigen Annahmen festlegt
- Programmiersprachen erlauben die Spezifikation syntaktischer Eigenschaften von Schnittstellen
- moderne Programmiersprachen bieten Schnittstellen als eigenes Sprachkonstrukt an
- nur wenige erlauben die Spezifikation semantischer Eigenschaften
  - Vor- und Nachbedingungen
  - weitere Zusicherungen, wie z.B. Speicher- und Zeitkomplexität

Paketdiagramme

Große Modelle benötigen Strukturierungen

Paketdiagramme

Zusammenfassung von Modellelementen zu Gruppen.

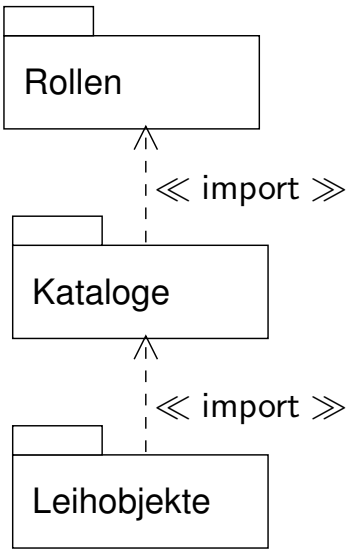
Anwendbarkeit:

- alle Arten von Modellelementen
- häufig für
  - Anwendungsfälle
  - Klassen (Programmiersprachenunterstützung durch Packages oder Namespaces)

Kriterien:

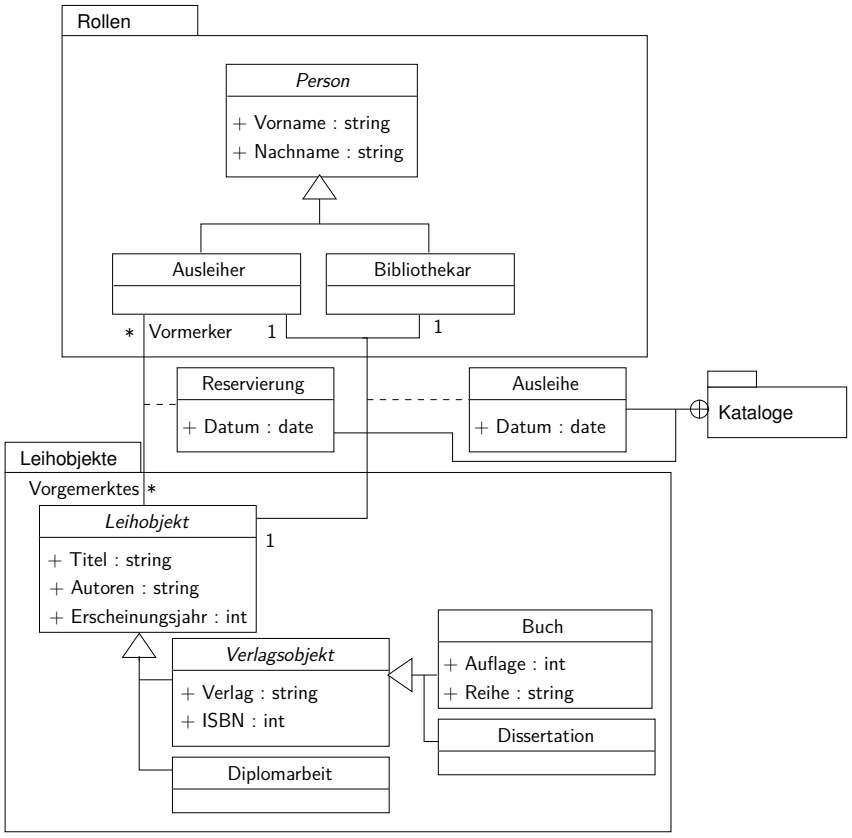
- logisch zusammengehörige Elemente
- 7 ± 3-Regel

Paketdiagramme



≪ import ≫: importierendes Paket fügt Namen des importierten Pakets zu eigenen Namen hinzu

Paketdiagramme: Schachtelung



Resultat der statischen Modellierung

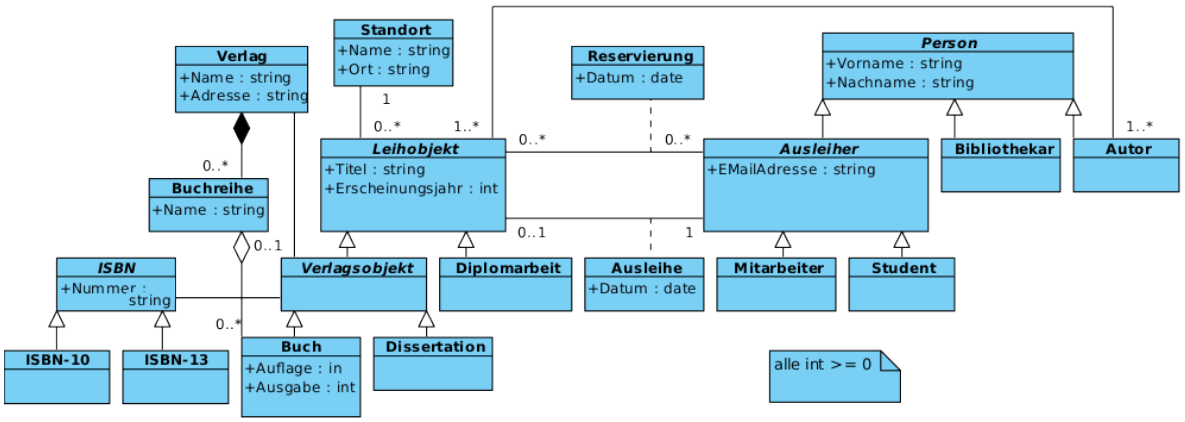
Definition

Statisches Datenmodell: beschreibt die statischen Konzepte und ihre Beziehungen.

- Domänenmodellierung: Konzepte stammen aus Anwendungsdomäne
- Entwurfsmodellierung: Konzepte sind Datenstrukturen für die Konzepte der Anwendungsdomäne und Konzepte aus der Programmierdomäne (z.B. Listen, Hashtabellen etc.)

Resultat der statischen Anforderungsanalyse im Beispiel

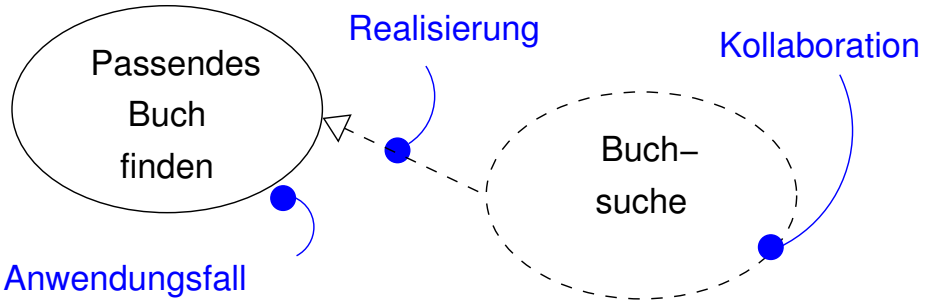
Datenmodell der Anwendungsdomäne:



Beschreibung von Anwendungsfällen

- textuelle Szenario-Beschreibungen (siehe oben)
- Aktivitätsdiagramme
- Interaktionsdiagramme
- Zustandsdiagramme

UML-Notation für Anwendungsfälle (OMG)



Anwendungsfälle können durch die inherit-Beziehung spezialisiert werden. Entspricht der objektorientierten Vererbung.

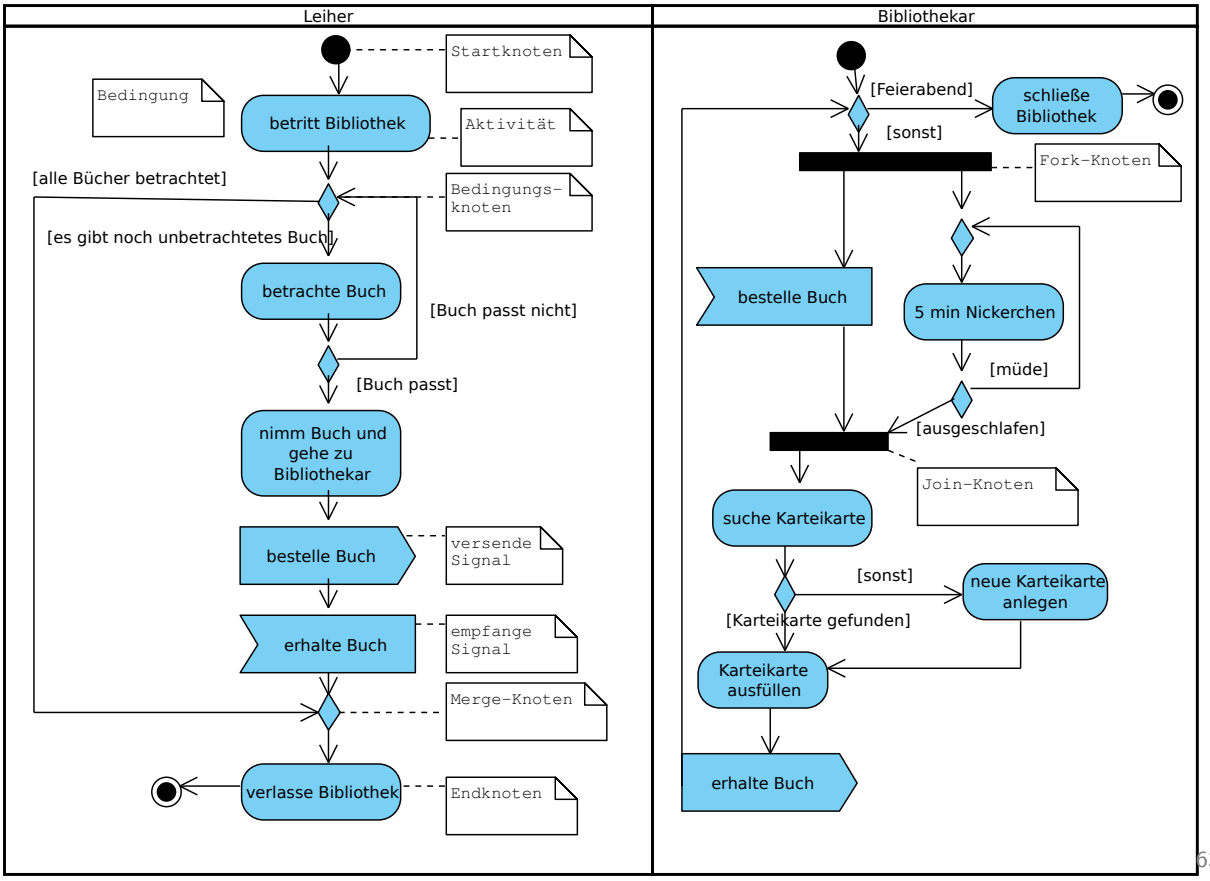
Aktivitätsdiagramme

- Wurzeln: Flussdiagramme, Petrinetze
- modellieren klassenübergreifendes Verhalten (Kontrollfluss)
- beschreiben häufig Anwendungsfälle
- Einsatz empfohlen bei hauptsächlich intern ausgelösten Zustandsübergängen (einfacher Kontrollfluss)

Laut UML-Standard 1.x ist ein Aktivitätsdiagramm der „Zustandsautomat einer Berechnung“. Ab UML 2.0 wird es auf Token-Konzept von Petri-Netzen zurück geführt.

Eine gute Darstellung findet man unter: `http://informatik.unibas.ch/uploads/media/se5full.pdf`

Geschäftsprozessmodellierung mit Aktivitätsdiagramm



Aktivitätsdiagramme: Bestandteile

- Aktionszustände: Zustände mit nur einer Eingangsaktion und ohne interne Transitionen
- Aufrufzustände: Aufruf einer Operation
- Subaktivitätszustände: Zustände mit internen Aktivitätsdiagrammen
- Entscheidungen („if —then—else“)
- Aktions-/Objekts-Flussbeziehungen
- Schwimmbahnen: unterteilen Zuständigkeiten für auszuführende Aktionen
- Kontroll-Icons für Signale

UML-Interaktionsdiagramme (OMG)

Beschreibungsinhalt:

- dynamische Systemaspekte
- exemplarische Beschreibung von Interaktionsabfolgen zwischen kollaborierenden Objekten (Inter-Objektverhalten)

zentrale Modellierungskonstrukte:

- Objekte: „Dinge“ mit eigener Identität
- Nachrichten
  - beschreiben den Austausch von Informationen zwischen Objekten zum Auslösen von Aktivitäten
  - sind Signale/Ereignisse oder Methodenaufrufe

UML-Interaktionsdiagramme (OMG)

Anwendung

- Präzisierung von Szenarien (exemplarische Folgen von Aktivitäten)
- Protokollierung des Nachrichtenaustausches
- Erhebung der von einzelnen Objekten bereit gestellten Funktionalität (Dienste, Methoden)

Varianten

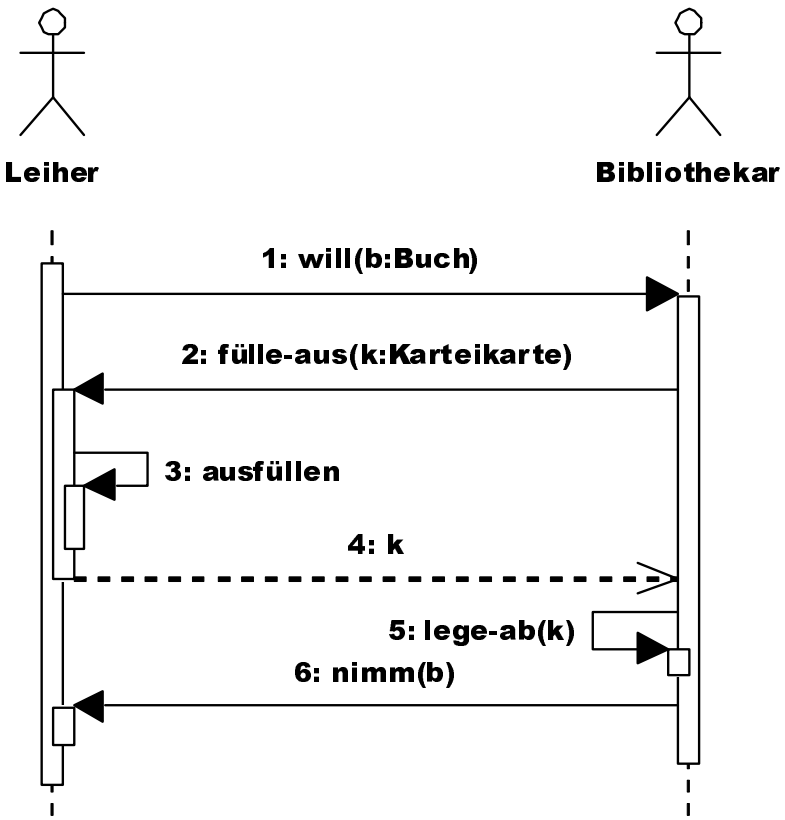
- Sequenzdiagramme
  - betonen zeitlichen Ablauf
- Kommunikationsdiagramme
  - betonen Objektstruktur

Grenzen:

- beschreiben Verhalten meist nur beispielhaft und unvollständig (nur Nachrichtenverkehr, nicht Inhalt)

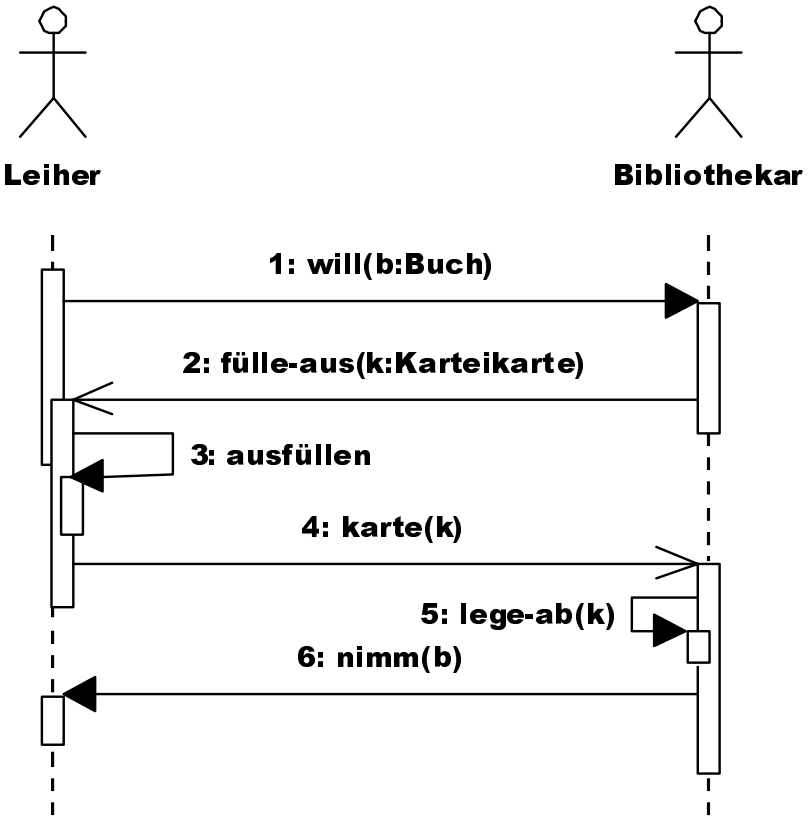
UML-Sequenzdiagramme (OMG)

Synchrone Kommunikation:



UML-Sequenzdiagramme (OMG)

Asynchrone Kommunikation:



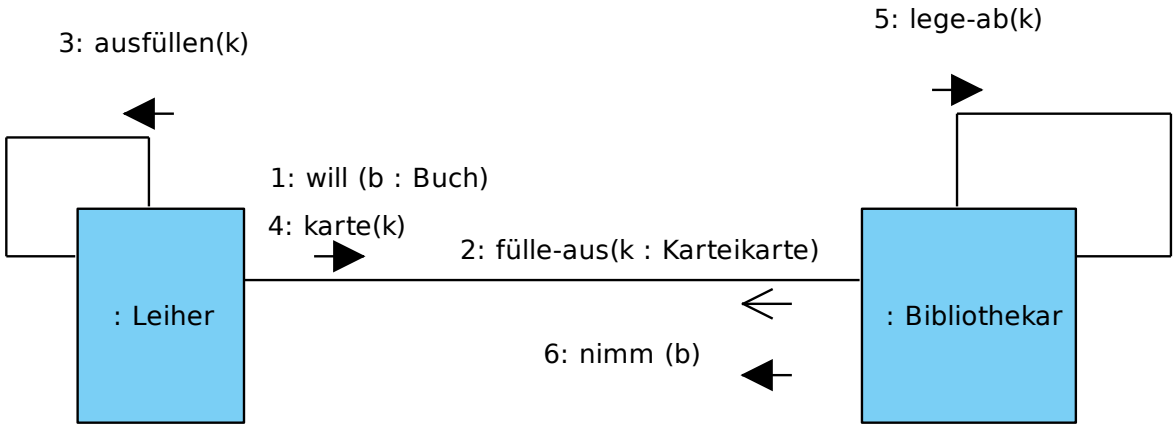
Aufruf:	synchrone Nachricht dargestellt durch durchgezogenen Pfeil vom Sender zum Empfänger mit gefüllter Pfeilspitze. Sender setzt Aktivität aus (gestrichelter Aktivierungsbalken); Empfänger wird aktiviert (Aktivierungsbalken beim Empfänger beginnt).
Antwort	synchrone Nachricht als Reaktion auf einen Aufruf dargestellt durch gestrichelter Pfeil vom Sender zum Empfänger mit offener Pfeilspitze). Antwortender Sender wird deaktiviert, ausgesetzter Empfänger lebt wieder auf.
Signal	asynchrone Nachricht dargestellt durch durchgezogenen Pfeil vom Sender zum Empfänger mit offener Pfeilspitze; Sender und Empfänger setzen ihre Aktivität parallel fort.

Bei synchroner Nachricht (Aufruf):

- Sender schickt Nachricht an Empfänger und gibt die Kontrolle über den weiteren Prozess an den Empfänger
- Empfänger gibt nach Bearbeitung der Nachricht die Kontrolle an den Sender zurück
- Interaktionen des Empfängers, die nach Erhalt der Nachricht begonnen wurden, müssen beendet sein, bevor der Empfänger die Kontrolle an den Sender zurückgibt (nested flow of control, method-call).

Anmerkung: Die Notation hat sich hier ab UML 2.0 geändert.

UML-Kommunikationsdiagramme (OMG)



Kommunikationsdiagramme

- beschreiben Interaktionen zwischen Objekten durch gerichtete Graphen
- die Abfolge der Nachrichten wird durch Nummerierung angezeigt

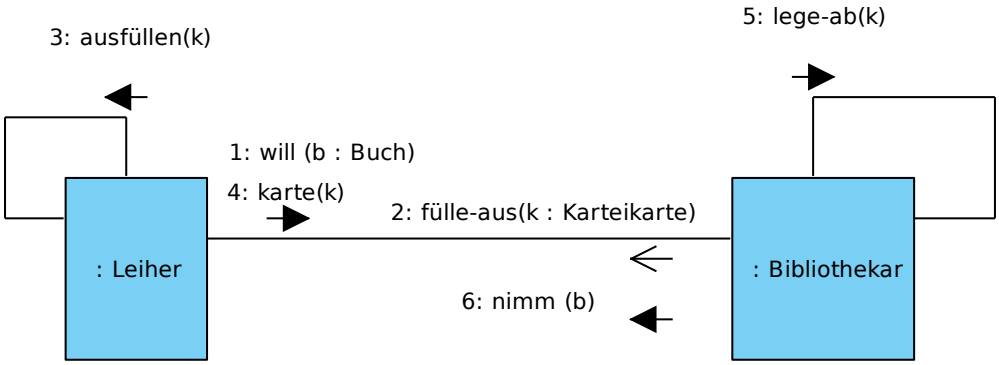
Sequenz- und Kommunikationsdiagramme beschreiben semantisch identische Inhalte mit unterschiedlichem Fokus (zeitlicher Ablauf bzw. Objekte stehen im Vordergrund).

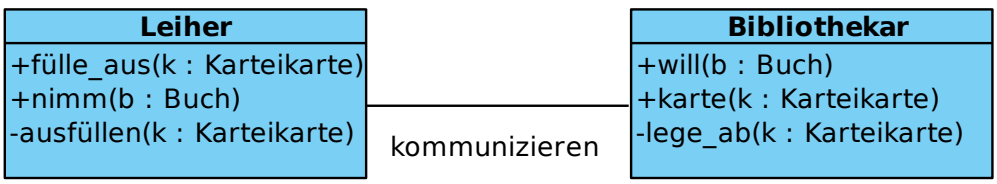
Interaktions- und Klassendiagramme

Bemerkung:

- Interaktionsdiagramme beschreiben exemplarische Interaktionsszenarien eines Systems aus dynamischer Sicht
- Klassendiagramme beschreiben diese Systeme aus statischer, schematischer Sicht
- Interaktionsdiagramme dienen als Ausgangspunkt zur Ergänzung und Überprüfung von Klassendiagrammen

Interaktionen → Methoden

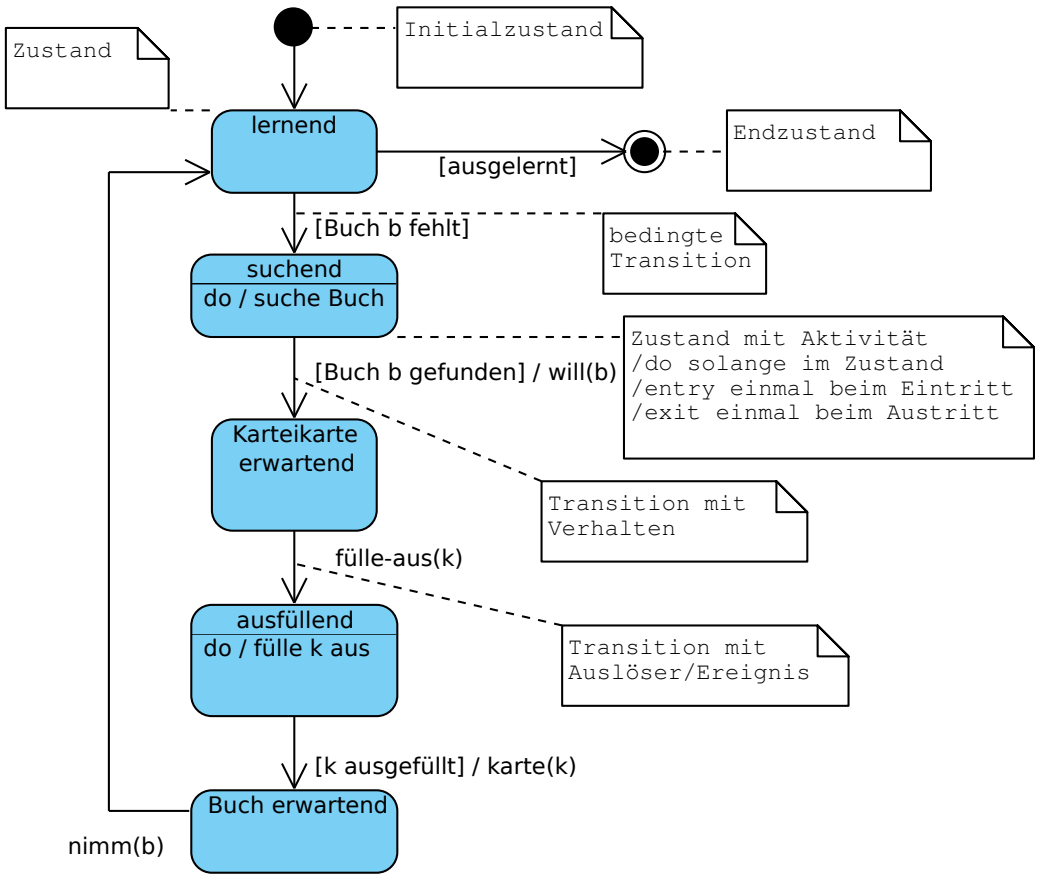




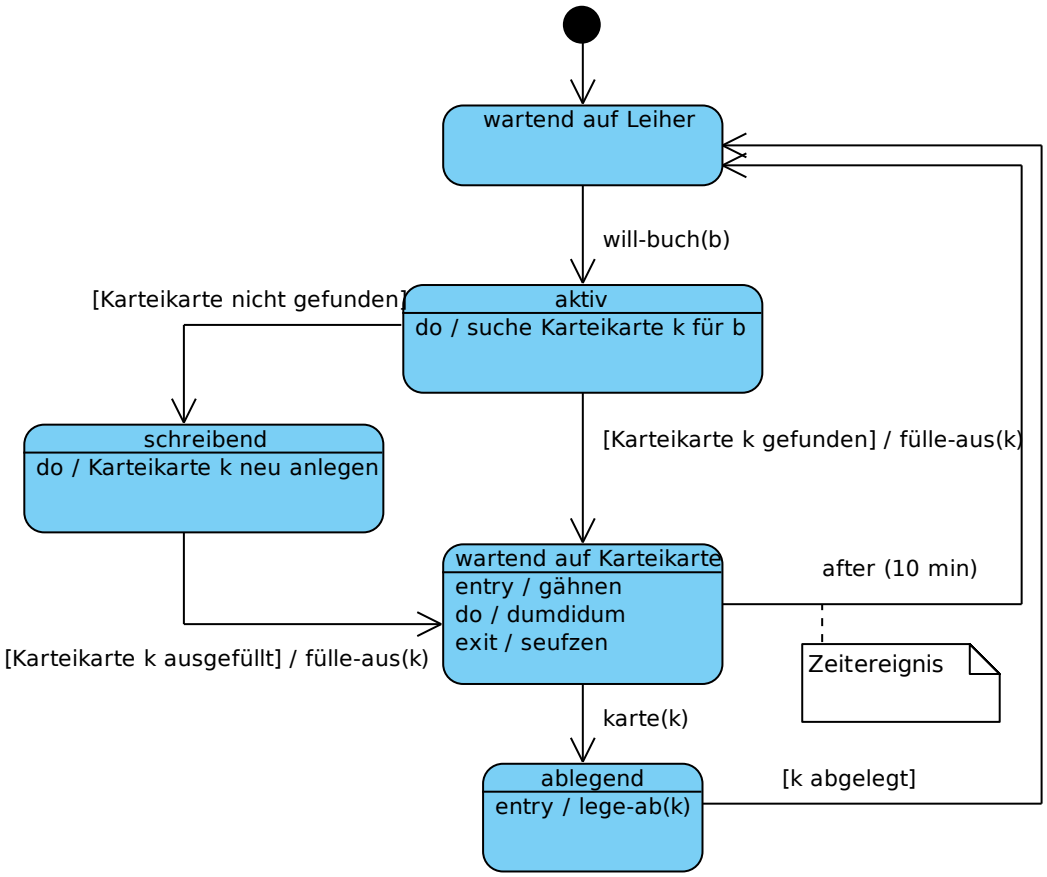
UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)
- **Zustand**: Menge von Attributwerten eines Objekts
- **Transition**: Zustandsänderung
- geeignet auch zur Beschreibung von Anwendungsfällen und Protokollen
- erlauben Verschachtelung von **Zuständen** und **Automaten**

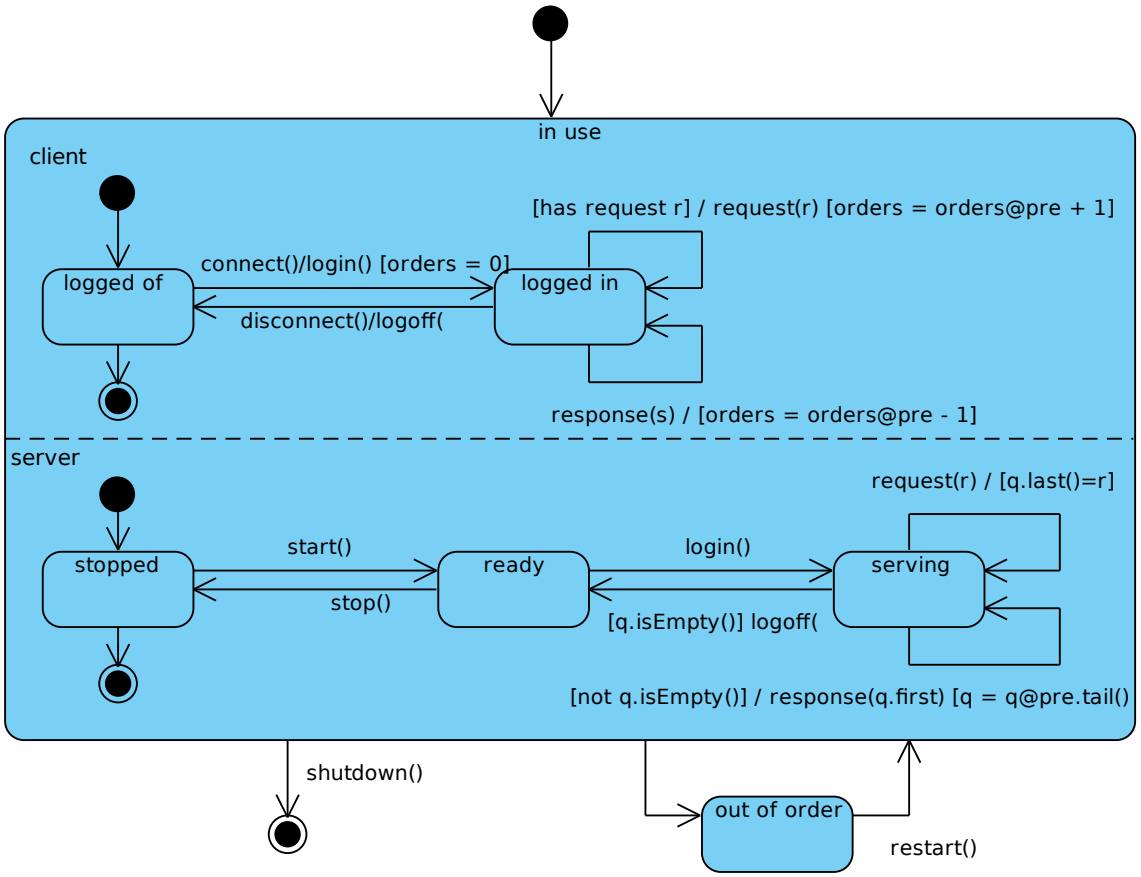
Zustandsautomaten für Ausleiher



Zustandsautomaten für Bibliothekar



Zustandsautomaten für Protokolle



```
1  class Server {
2
3      private enum States {initial, stopped, ready, serving};
4      private States state;
5      private Queue requests;
6
7      public Server() {state = States.stopped; }
8      public void start() throws Exception {
9          if (state == States.stopped)
10             state = States.ready;
11         else throw new Exception();
12     }
13     public void stop() throws Exception  {
14         if (state == States.ready)
15             state = States.stopped;
16         else throw new Exception();
17     }
18     public void login() throws Exception  {
19         if (state == States.ready)
20             state = States.serving;
21         else throw new Exception();
22     }
23     void logoff() throws Exception  {
24         if (state == States.serving) {
25           if (requests.isEmpty()) {
26             state = States.ready;
27           } else throw new Exception();
28         } else throw new Exception();
29     }
30     public void request(int r) throws Exception  {
31         if (state == States.serving) {
32                 requests.add(r); assert (requests.last() == r);
33         } else throw new Exception();
34     }
35
```



```
36
37     public int response() throws Exception  {
38         if (state == States.serving) {
39             if (!requests.isEmpty()) {
40                 int r = requests.first(); requests = requests.tail();
41                 return r;
42             } else throw new Exception();
43         } else throw new Exception();
44     }
45 };
```

Zustände

- einfach/zusammengesetzt
- „interne“ Aktionen: entry, exit, do, include, frei definierbare Events

Zusammengesetzte Zustände

- mehrere nebenläufige **Regionen** möglich
- Parallelität durch Gabelung/Join (bedingungsfrei!)

Klauseln im Zustand:

- do/ activity : Aktivität, die im Zustand ausgeführt wird
- entry/ action : bei Eintritt ausgeführte Aktivität
- exit/ action : bei Austritt ausgeführte Aktivität
- include/ : Zustandsmaschine wird importiert

Transitionen

- ereignisabhängig:
  Event [Precondition] / Trigger^Send Event [Postcondition]
- Auslöser (Trigger): Ereignis, Bedingung, Zeit
- Auswirkung: Aktion (z.B. Methodenaufruf), Zustandsänderung, Nachbedingung (Prädikat, das nach Transition gilt)

Anwendung von Zustandsdiagrammen

- Beschreibung von Objektlebenszyklen pro Klasse
- Beschreibung von Protokollen
- Verfeinerung von Anwendungsfällen

Interaktionen versus Methoden

Interaktionen

- beschreiben das Wechselspiel zwischen Akteuren über Botschaften/Methoden[1]
- liefern Methoden für die modellierten Klassen
- beschreiben jedoch **nicht** die Methoden selbst

[1] Methoden beschreiben hier abstraktes Verhalten im Kontext der Anforderungen; sind noch nicht notwendigerweise die Methoden im Sinne der Implementierung (führen letztlich aber zu diesen hin).

Operationen/Nachrichten

Beschreibung:

- **Parameter:** Eingabe und Ausgabe
- **Vorbedingung:** beschreibt die Annahmen der Methode, die gelten müssen, damit sie ausgeführt werden kann
- **Nachbedingung:** beschreibt das Resultat der Methode
- **Fehlerbedingungen:** Verletzung der Vorbedingungen und Fehler, die während der Ausführung auftreten können
- **Verhalten in Fehlersituationen:** Nachbedingung für jeden aufgetretenen Fehler
- **Reaktionszeit:** Maximale Dauer, bis Resultat vorliegt (sowohl im Normal- als auch im Fehlerfall)

Operationen/Nachrichten

Beispiel Sortierung und Ausgabe der Buchliste:

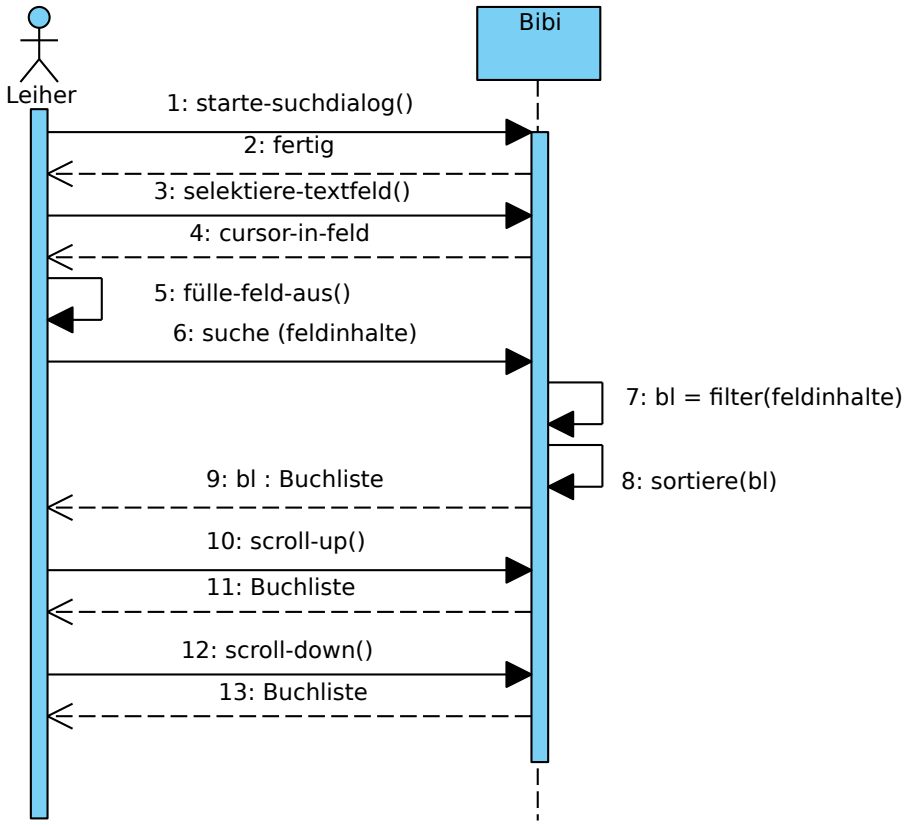
- **Parameter:**
  - Eingabe: Buchliste, Attribut
  - Ausgabe: Buchliste'
- **Vorbedingung:**
  - Attribut kommt bei allen Büchern der Buchliste vor
- **Nachbedingung:**
  - Buchliste' ist sortiert, d.h.:
    $\forall i : 1 \leq i < len(Buchliste')$
    $\Rightarrow element(Buchliste', i) \leq_{Attribut} element(Buchliste', i+1)$
  - Buchliste' ist eine Permutation von Buchliste
- **Fehlerbedingungen:** keine, außer Vorbedingung nicht erfüllt
- **Verhalten in Fehlersituationen:**
  - Buchliste wird zurückgegeben
  - Fehlermeldung wird ausgegeben
- **Reaktionszeit:** $n \cdot log(n) \cdot 0,001$ [sec], wobei n die Länge der Liste ist

Fragen

Wie lässt sich graphische Benutzeroberfläche modellieren?
Aspekte:

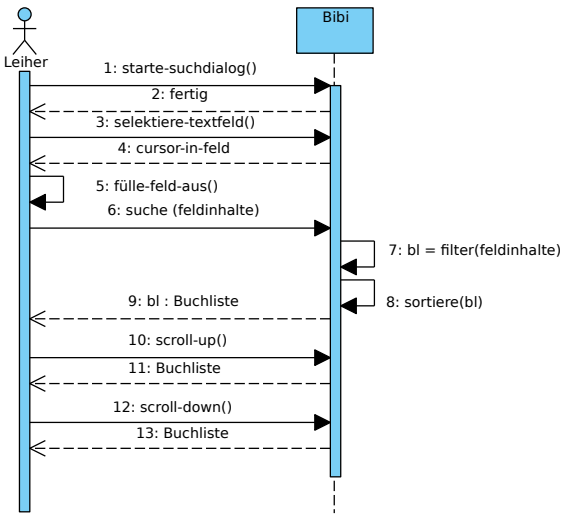
- Verhalten
- Darstellung

Interaktion zwischen Benutzer und System (Dialog)



Zusammenhang von Benutzernachrichten und GUI

Nachrichten vom Benutzer: Ereignisse der Eingabegeräte



selektiere-textfeld ()
: linker Mausklick in Textfeld

fülle-feld-aus ()
: Tastatureingabe

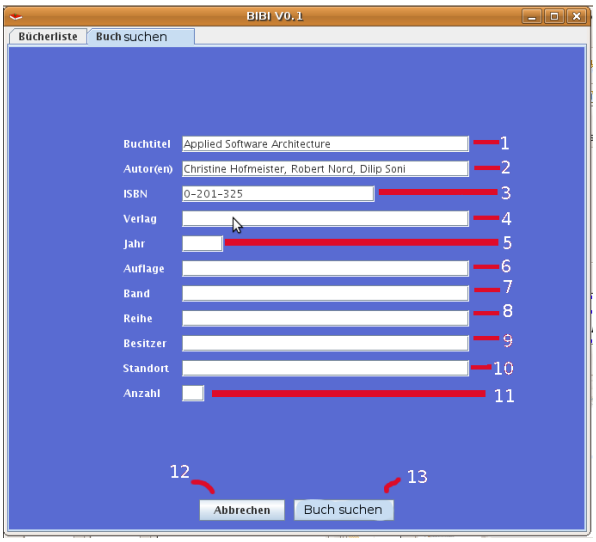
suche (feldinhalte)
: linker Mausklick auf Schaltfläche “Buch suchen”

scroll-up ()
: Scrollbar nach oben schieben oder Mausrad nach vorne

scroll-down ()
: Scrollbar nach unten schieben oder Mausrad nach unten

Modellierung der Darstellung

Nachrichten an Benutzer: Anzeige in GUI (hier: Suchdialog)



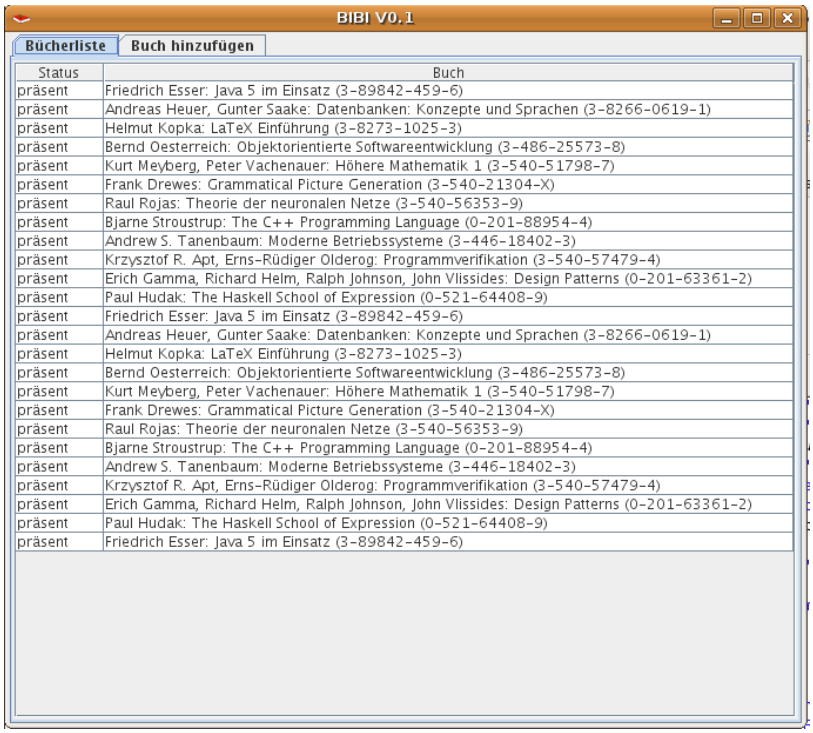
1-11 Buchdaten (10pt-Schrift)

12 Schaltfläche fürs Abbrechen

13 Schaltfläche fürs Starten der Suche

Modellierung der Darstellung

Resultat der Suche



Modellierung von Benutzerschnittstellen

Darstellung

- Papierskizzen
- Bilder
- Screenshots
- Animierte Darstellungen

Verhalten

- Interaktionsdiagramme
- Zustandsdiagramme, z.B.:
  - Dialoge $\rightarrow$ zusammengesetzte Zustände
  - Interna der Dialoge $\rightarrow$ Interna in zusammengesetzten Zuständen
  - Übergänge zwischen Dialogen $\rightarrow$ Transitionen zwischen zusammengesetzten Zuständen

Interaktiver Prototyp beinhaltet all dies als Referenzimplementierung.

Wiederholungsfragen I

- Was ist ein Modell? Wann und wozu modellieren wir?
- Welches sind die wesentlichen Schritte der objektorientierten Modellierung im Kontext der Anforderungsspezifikation?
- Was ist ein Anwendungsfall? Wie können Sie Anwendungsfälle beschrieben?
- Was ist ein Akteur?
- Welche Beziehungen zwischen Anwendungsfällen unterstützt die UML?
- Was ist der Unterschied zwischen Objekt- und Klassendiagrammen?
- Erläutern Sie Klassendiagramme für die Datenmodellierung.
- Was ist eine Generalisierung?
- Worin besteht das Liskovsche Substitutionsprinzip?
- Wie lassen sich attributierte Assoziationen und n-stellige Assoziationen mit $n > 2$ darstellen?

Wiederholungsfragen II

- Was ist eine Aggregation bzw. eine Komposition?
- Wie wird ein Anwendungsfall textuell beschrieben?
- Erläutern Sie Interaktionsdiagramme (Sequenzdiagramme und Kommunikationsdiagramme).
- Erstellen Sie ein Sequenzdiagramm für ein bestimmtes Szenario.
- Überführen Sie dieses Sequenzdiagramm in ein Kommunikationsdiagramm.
- Wie lassen sich Operationen aus den Interaktionsdiagrammen ableiten?
- Wie sind Operationen zu beschreiben?
- Erläutern Sie Aktivitätsdiagramme der UML. Erstellen Sie ein Aktivitätsdiagramm für ein bestimmtes Szenario.
- Erläutern Sie Zustandsautomatendiagramme der UML. Erstellen Sie ein Zustandsautomatendiagramm für ein bestimmtes Szenario.
- Wofür können die genannten Diagrammarten benutzt werden?

Weiterführende Literatur/Links

Buchtipps: Störrle (2005) und Rupp u. a. (2007)

Ein kurzes Tutorial in Deutsch:
`http://ivs.cs.uni-magdeburg.de/~dumke/UML/`

Eine sehr kurze Übersicht in Englisch:
`http://bdn.borland.com/article/0,1410,31863,00.html`

Weitere ausführlichere Tutorials in Englisch:
`http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/`
`http://uml.tutorials.trireme.com/`

1 Harel 1987 Harel, David: State Charts: a Visual Formalism for Complex Systems. In: Science of Computer Programming 8 (1987), Nr. 3, S. 231–274

2 Liskov 1988 Liskov, Barbara: Data Abstraction and Hierarchy. In: SIGPLAN Notices 23 (1988), Mai, Nr. 5

3 Liskov und Wing 1994 Liskov, Barbara ; Wing, Jeanette M.: A Behavioral Notion of Subtyping. In: ACM Transactions on Programming Lnguages and Systems 16 (1994), Nr. 6, S. 1811–1841

4 OMG Object Management Group (OMG): Unified Modeling Language. `http://www.uml.org`

5 Parnas 1972 Parnas, David L.: On the Criteria to Be Used in Decomposing Systems into Modules. In: Communications of the ACM 15 (1972), Dezember, Nr. 12

6 Rupp u. a. 2007 Rupp, Chris ; Queins, Stefan ; Zengler, Barbara: UML 2 glasklar. 3. Auflage. Hanser Verlag, 2007. – ISBN 978-3-446-41118-0

7 Störrle 2005 Störrle, Harald: UML 2 für Studenten. Pearson Studium, 2005. – ISBN 3-8273-7143-0